Data Mining

National Crime Victimization Survey

Varun Nadgir Vijay Kesari

Data Introduction

Before loading any data into R, our first step was to read the provided NCVS codebook and become familiar with the context of the study. This allowed us to build an understanding of the types of responses and what we would need to be mindful of in the dataset. We found that the responses are all categorical-for example, the field V2022 is for "Location of Phone" and has possible values ranging from 1 to 9. According to the codebook, the responses will fall into one of the following categories:

- Valid values are usually in the range of smaller integer numbers.
- Explicit "don't know" responses are ones that are clearly offered as a response type
- Blind "don't know" responses are not offered but still accepted for most questions
- Refused response means the respondent chose not to answer
- "Residue" responses usually mean something is missing and are usually identified by '8', '98', '99', '998', etc. based on the length of an accepted response
- "Out of scope/universe" responses are those that are not applicable to the question

There are several reasons why a response might be blank, causing some ambiguity between blind "don't know" responses and refused responses. However, we can handle residue and out of scope responses in our preprocessing.

Our initial dataset contains 4947 tuples of 204 variables before any preprocessing. In addition to the 203 training variables, the target variable is *o_bullied* - it has a value of 1 if the student experienced bullying and 0 if they did not. Because we have a positive and negative class, we will be building binary classifiers.

Data Mining Tools

- modeest
- dplyr
- e1071
- caret
- rsample
- rpart
- randomForest
- xgboost
- glmnet
- pROC
- class
- nnet

Preprocessing

Since there are results that indicate non-answers, we will be avoiding including those in our training. There are also several columns that are heavily populated by a single response. Our first step is to target low variance and noisy data. One way to address these in bulk is to simply filter by number of factors. Variables with only a few levels will not contain Residual or Out of universe responses

Classification Algorithms

- Algorithm 1: Logistic Regression
- Algorithm 2: Decision Tree/Random Forest
- Algorithm 3: Gradient Boost (XGBoost)

- Algorithm 4: Naive Bayes
- Algorithm 5: K-Nearest Neighbors
- Algorithm 6: Neural Network
- Algorithm 7: SVM

Data Mining Procedure

```r
library(modeest)
library(dplyr)
library(e1071)
library(caret)
library(rsample)
library(rpart)
library(randomForest)
library(xgboost)
library(glmnet)
library(pROC)
library(class)
library(nnet)

# 1) import initial data
data <- read.csv("project_dataset.csv")
```

First, to reduce the number of Residue and Out of universe responses, we realized we could not universally remove 8's and 9's from our data. Some fields have beyond 9 options, meaning that the values 8 and 9 are valid responses. Thinking about it logically, we concluded that if a field does not contain 7 levels, then the responses of 8 or 9 *cannot* be valid, as there are not enough options. By turning these to NA, we have filtered out many of the invalid responses, though not all. Then, by setting a threshold for the number of acceptable NA values a field can have, we can further reduce the number of columns.

```r
# 2) list of columns containing under 7 levels
level_below_7 <- sapply(data, function(col) nlevels(col) < 7)

# 3) if a column does not contain at least 7 levels, then values of 8 and 9 MUST be invalid
data[, !level_below_7] <- lapply(data[, !level_below_7], function(x) ifelse(x >= 8, NA, x))

# 4) replace values greater than or equal to 98 with NA (out of universe)
data[data>=9] <- NA

# 5) set a threshold for number of NA values in a column
na_thresh <- 3000

# drop columns that have NA values over our threshold
columns_to_remove <- colSums(is.na(data)) > na_thresh

# this reduces us down to 153 training variables
data <- data[, !columns_to_remove]

# 6) apply mfv from modeest package on every column
data <- data %>%
  mutate_all(~ ifelse(is.na(.), mfv(., na_rm=TRUE), .))
```

```r
# 7) identify columns near zero variance (heavily one level)
near_zero_var <- nearZeroVar(data[1:length(data)-1], freqCut=85/15)

# this reduces us down to 74 training variables
data <- data[ ,-near_zero_var]
```

After, we change all the columns to be factor, so that they are treated as having levels. By filtering for a given number of levels, such as 2 or 3, we are able to test and debug our models at low runtime cost and then gradually scale up. We then create the dummy variables and encode the dataset, which expands the number of columns we have but allows us to use numerical analysis on our values.

```r
# 8) convert all variables to factors
data <- as.data.frame(lapply(data, as.factor))

# 9) filter factor variables with given number of levels
data_filtered <- data[, sapply(data, function(col) length(levels(col)) %in% c(2,3,4,5))]

# 10) create dummy vars and encode
data_dummy <- dummyVars("~ .", data = data_filtered, fullRank = TRUE)
data_encoded <- as.data.frame(predict(data_dummy, newdata = data_filtered))
names(data_encoded)[names(data_encoded) == "o_bullied.1"] <- "o_bullied"

# set seed for reproducibility
set.seed(123)

# 11) split 70-30
split <- initial_split(data_encoded, prop=0.70)
tr_encoded <- training(split)
ts_encoded <- testing(split)

# 12) remove columns of 0-only, whether in tr or ts
all_zero_col <- c(unname(which(sapply(tr_encoded, function(col) all(col==0)))),
                  unname(which(sapply(ts_encoded, function(col) all(col==0)))))
all_zero_col <- unique(all_zero_col)

if (length(all_zero_col)>0) {
  tr_encoded <- tr_encoded[, -all_zero_col]
  ts_encoded <- ts_encoded[, -all_zero_col]
}

# show proportion of target variable in all datasets
table(data_encoded$o_bullied)/length(data_encoded$o_bullied)
```

```
##
##         0         1
## 0.7715787 0.2284213
```

```r
table(tr_encoded$o_bullied)/length(tr_encoded$o_bullied)
```

```
##
##         0         1
## 0.7732525 0.2267475
```

```r
table(ts_encoded$o_bullied)/length(ts_encoded$o_bullied)
```

```
##
##         0         1
## 0.7676768 0.2323232
```

Now that we have our main dataset and the train and test set, we can check the proportion of class 0 to class 1. As we can see, it is skewed towards the negative class, or N, which may pose issues when it comes to training and predicting, since our model will have seen fewer class 1 in training. If necessary, we can attempt to over- or under-sample our data to deal with the imbalance.

Model Results & Evaluation

By creating a function to calculate the performance measures based on a given confusion matrix, we can keep our calculations clean. By using class weights as function arguments, we can adjust as needed for each model we train, giving us flexibility. Additionally, we have a function for identifying factor levels that are not present in both tr and ts, since this may happen for some splits. See **Appendix 1.1** for the full *calculate_measures* function.

Model 1: Logistic Regression

```r
# 5-fold cv
ctrl <- trainControl(method="cv", number=5)

# train model
m <- train(o_bullied ~ ., data = tr_encoded, method = "glm",
           family = "binomial", trControl = ctrl)

# stop scientific notation
options(scipen=999)

# compute class probabilities
m.pred <- predict(m, ts_encoded[, -length(ts_encoded)])

# change 0,1 to N,Y and set as factor
pred <- factor(ifelse(m.pred>=0.3, "Y", "N"))
target <- factor(ifelse(ts_encoded$o_bullied==1, "Y", "N"))

# create confusion matrix with Y as positive class
cm <- confusionMatrix(data=pred, reference=target, positive="Y")

# use default weights 1:1
measures <- calculate_measures(cm$table)
```

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class N	0.8132	0.4551	0.8552	0.8132	0.8336	0.679	0.3406	0.7508
Class Y	0.5449	0.1868	0.4688	0.5449	0.504	0.679	0.3406	0.7508
Avg.	0.679	0.6605	0.662	0.679	0.6688	0.679	0.3406	0.7508

This model appears to slightly favor predicting the negative class, which we expected might be an issue. To help with this, we can introduce regularization to our logistic regression model to reduce some overfitting. Additionally, we can consider lowering the threshold for classification to see if having more accurate positive TPR has a severe, negative tradeoff.

```r
# set target back to numeric
tr_encoded$o_bullied <- as.numeric(as.character(tr_encoded$o_bullied))

# create training/response matrix
X <- model.matrix(o_bullied ~ ., data = tr_encoded)[, -1]
y <- tr_encoded$o_bullied

# cv grid
cv <- cv.glmnet(X, y, alpha = 1)

# train model with optimal lambda
m <- glmnet(X, y, alpha = 1, lambda = cv$lambda.min)

# Make predictions on the test set
ts_X <- model.matrix(o_bullied ~ ., data = ts_encoded)[, -1]
m.pred <- predict(m, newx = ts_X, s = cv$lambda.min, type = "response")

# change 0,1 to N,Y and set as factor
pred <- factor(ifelse(m.pred >= 0.25, "Y", "N"))
target <- factor(ifelse(ts_encoded$o_bullied==1, "Y", "N"))

# create confusion matrix with Y as positive class
cm <- confusionMatrix(data=pred, reference=target, positive="Y")

# use default weights 1:1
measures <- calculate_measures(cm$table)
```

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class N	0.7114	0.3362	0.8749	0.7114	0.7847	0.6876	0.3271	0.7003
Class Y	0.6638	0.2886	0.4104	0.6638	0.5072	0.6876	0.3271	0.7003
Avg.	0.6876	0.6562	0.6426	0.6876	0.646	0.6876	0.3271	0.7003

It seems that lowering the threshold and adding regularization has helped quite a bit, as TPR and FPR for both classes has in fact improved.

Model 2: Decision Tree / Random Forest

```r
# train the decision tree model
m <- rpart(o_bullied ~ ., data = tr_encoded)

# predict probabilities for the positive class
m.pred <- predict(m, ts_encoded[, -length(ts_encoded)])

# change 0,1 to N,Y and set as factor
pred <- factor(ifelse(m.pred >= 0.5, "Y", "N"))
```

```r
target <- factor(ifelse(ts_encoded$o_bullied == 1, "Y", "N"))

# create confusion matrix with Y as the positive class
cm <- confusionMatrix(data = pred, reference = target, positive = "Y")

# use default weights 1:1
measures <- calculate_measures(cm$table)
```

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class N	0.9746	0.858	0.7896	0.9746	0.8724	0.5583	0.2207	0.7811
Class Y	0.142	0.0254	0.6282	0.142	0.2317	0.5583	0.2207	0.7811
Avg.	0.5583	0.7209	0.7089	0.5583	0.552	0.5583	0.2207	0.7811

This model behaves even worse than the first - its predictions heavily favor the N class. We can instead use a random forest and adjust the threshold once again.

```r
# train random forest
m <- randomForest(o_bullied ~ ., data=tr_encoded, ntree=100)

m.pred <- predict(m, ts_encoded[ , -length(ts_encoded)])

# change 0,1 to N,Y and set as factor
pred <- factor(ifelse(m.pred>=0.26, "Y", "N"))
target <- factor(ifelse(ts_encoded$o_bullied==1, "Y", "N"))

# create confusion matrix with Y as positive class
cm <- confusionMatrix(data=pred, reference=target, positive="Y")

# use default weights 1:1
measures <- calculate_measures(cm$table)
```

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class N	0.6693	0.287	0.8852	0.6693	0.7622	0.6912	0.3272	0.6795
Class Y	0.713	0.3307	0.3949	0.713	0.5083	0.6912	0.3272	0.6795
Avg.	0.6912	0.6544	0.64	0.6912	0.6353	0.6912	0.3272	0.6795

By giving up TPR in the negative class, we get quite a bit of improvement in the positive TPR.

Model 3: Gradient Boost / XGBoost

```r
# create xgb matrix
xgb_matrix <- xgb.DMatrix(data=as.matrix(tr_encoded[ , -length(tr_encoded)]),
                          label=tr_encoded$o_bullied)

# train binary model
m <- xgboost(data=xgb_matrix, nrounds=100, objective="binary:logistic", verbose=0)
```

```r
# predict and evaluate
xgb_predictions <- predict(m, as.matrix(ts_encoded[ , -length(ts_encoded)]))
xgb_predictions <- ifelse(xgb_predictions >= 0.2, "Y", "N")
xgb_predictions <- as.factor(xgb_predictions)
xgb_cm <- confusionMatrix(data=xgb_predictions, reference=target, positive="Y")
measures <- calculate_measures(xgb_cm$table)
```

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class N	0.764	0.4841	0.8391	0.764	0.7998	0.64	0.2578	0.7064
Class Y	0.5159	0.236	0.3982	0.5159	0.4495	0.64	0.2578	0.7064
Avg.	0.64	0.68	0.6187	0.64	0.6247	0.64	0.2578	0.7064

This model performs decently, but the threshold started to trend too low and we were hesitant to set it too low.

Model 4: Naive Bayes

```r
m <- cv.glmnet(as.matrix(tr_encoded[, -length(tr_encoded)]), tr_encoded$o_bullied,
               family = "binomial", alpha = 1, nfolds = 10)

# Predict class probabilities
m.pred <- predict(m, newx = as.matrix(ts_encoded[, -length(ts_encoded)]),
                  s = "lambda.min", type = "response")

# Convert probabilities to class labels
pred <- factor(ifelse(m.pred >= 0.23, "Y", "N"))
target <- factor(ifelse(ts_encoded$o_bullied == 1, "Y", "N"))

# Create confusion matrix with Y as the positive class
cm <- confusionMatrix(data = pred, reference = target, positive = "Y")

# Calculate evaluation measures
measures <- calculate_measures(cm$table)
```

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class N	0.714	0.3304	0.8772	0.714	0.7872	0.6918	0.3346	0.7037
Class Y	0.6696	0.286	0.4147	0.6696	0.5122	0.6918	0.3346	0.7037
Avg.	0.6918	0.6541	0.6459	0.6918	0.6497	0.6918	0.3346	0.7037

This is another good model that performs as well as our second logistic regression model.

Model 5: KNN

```r
m.pred <- knn(train = tr_encoded,
              test = ts_encoded,
              cl = tr_encoded[, length(tr_encoded)],
              k = 5)

# Convert class labels to N/Y
pred <- factor(ifelse(m.pred==1, "Y", "N"))
target <- factor(ifelse(ts_encoded$o_bullied == 1, "Y", "N"))

# Create confusion matrix with Y as the positive class
cm <- confusionMatrix(data = pred, reference = target, positive = "Y")

measures <- calculate_measures(cm$table)
```

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class N	0.9939	0.8058	0.803	0.9939	0.8883	0.594	0.365	0.8081
Class Y	0.1942	0.0061	0.9054	0.1942	0.3198	0.594	0.365	0.8081
Avg.	0.594	0.703	0.8542	0.594	0.604	0.594	0.365	0.8081

Model 6: Neural Network

```
# Neural network parameters
size <- 4  # Number of units in the hidden layer
decay <- 0.1  # Weight decay (regularization term)

train_data <- tr_encoded[, -length(tr_encoded)]
test_data <- ts_encoded[, -length(ts_encoded)]
train_labels <- as.numeric(tr_encoded[, length(tr_encoded)])
test_labels <- as.numeric(ts_encoded[, length(ts_encoded)])

# Fit the neural network model
nn_model <- nnet(train_data, train_labels, size = size, decay = decay, maxit = 200, trace=FALSE)

# Predict on the test set
nn_predictions <- predict(nn_model, test_data)

# Convert class labels to N/Y
pred <- factor(ifelse(nn_predictions>=0.23, "Y", "N"))
target <- factor(ifelse(ts_encoded$o_bullied == 1, "Y", "N"))

# Create confusion matrix with Y as the positive class
cm <- confusionMatrix(data = pred, reference = target, positive = "Y")

measures <- calculate_measures(cm$table)
```

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class N	0.7711	0.4812	0.8411	0.7711	0.8046	0.6449	0.2681	0.7125
Class Y	0.5188	0.2289	0.4068	0.5188	0.4561	0.6449	0.2681	0.7125
Avg.	0.6449	0.6775	0.624	0.6449	0.6303	0.6449	0.2681	0.7125

Model 7: Support Vector Machine (SVM)

```r
train_data_scaled <- tr_encoded[, -length(tr_encoded)] # excluding the target variable
test_data_scaled <- ts_encoded[, -length(ts_encoded)]  # excluding the target variable

train_labels <- factor(tr_encoded[, length(tr_encoded)])
test_labels <- factor(ts_encoded[, length(ts_encoded)])

# Train an SVM model on the training data
svm_model <- svm(train_data_scaled, train_labels, kernel = "radial", cost = 1, scale = TRUE)

# Predict using the SVM model on the test data
svm_predictions <- predict(svm_model, test_data_scaled)

# Convert probabilities to class labels based on the threshold
# Convert probabilities to class labels
pred <- factor(ifelse(svm_predictions == 1, "Y", "N"))
target <- factor(ifelse(ts_encoded$o_bullied == 1, "Y", "N"))

# Create confusion matrix with Y as the positive class
cm <- confusionMatrix(data = pred, reference = target, positive = "Y")

# Calculate evaluation measures
measures <- calculate_measures(cm$table)
```

	TPR	FPR	Precision	Recall	F-measure	ROC	MCC	Kappa
Class N	0.9798	0.9333	0.7762	0.9798	0.8662	0.5232	0.1133	0.7677
Class Y	0.0667	0.0202	0.5	0.0667	0.1176	0.5232	0.1133	0.7677
Avg.	0.5232	0.7384	0.6381	0.5232	0.4919	0.5232	0.1133	0.7677

Appendix

1.1:

```r
# define function to calculate performance measures based on given confusion matrix
calculate_measures <- function(cm, N_weight=1, Y_weight=1) {
  # target Positive Class
  tp_pos <- cm[2, 2]
  fp_pos <- cm[2, 1]
  tn_pos <- cm[1, 1]
  fn_pos <- cm[1, 2]

  # target Negative Class
  tp_neg <- cm[1, 1]
  fp_neg <- cm[1, 2]
  tn_neg <- cm[2, 2]
  fn_neg <- cm[2, 1]

  # calculate all Positive measures
  tpr_pos <- tp_pos/(tp_pos+fn_pos)
  fpr_pos <- fp_pos/(fp_pos+tn_pos)
  precision_pos <- tp_pos/(tp_pos+fp_pos)
  recall_pos <- tpr_pos
  f_measure_pos <- (2*precision_pos*recall_pos)/(precision_pos+recall_pos)
  roc_auc_pos <- (1+tpr_pos-fpr_pos)/2
  mcc_pos <- (tp_pos*tn_pos-fp_pos*fn_pos)/
    (sqrt(tp_pos+fp_pos)*sqrt(tp_pos+fn_pos)*sqrt(tn_pos+fp_pos)*sqrt(tn_pos+fn_pos))
  k_pos <- (tp_pos+tn_pos)/(tp_pos+tn_pos+fp_pos+fn_pos)

  # calculate all Negative measures
  tpr_neg <- tp_neg/(tp_neg+fn_neg)
  fpr_neg <- fp_neg/(fp_neg+tn_neg)
  precision_neg <- tp_neg/(tp_neg+fp_neg)
  recall_neg <- tpr_neg
  f_measure_neg <- (2*precision_neg*recall_neg)/(precision_neg+recall_neg)
  roc_auc_neg <- (1+tpr_neg-fpr_neg)/2
  mcc_neg <- (tp_neg*tn_neg-fp_neg*fn_neg)/
    (sqrt(tp_neg+fp_neg)*sqrt(tp_neg+fn_neg)*sqrt(tn_neg+fp_neg)*sqrt(tn_neg+fn_neg))
  k_neg <- (tp_neg+tn_neg)/(tp_neg+tn_neg+fp_neg+fn_neg)

  # calculate weighted averages
  weights <- c(N_weight, Y_weight)

  tpr_avg <- weighted.mean(c(tpr_neg, tpr_pos), weights)
  fpr_avg <- weighted.mean(c(fpr_neg, fpr_pos, weights))
  precision_avg <- weighted.mean(c(precision_neg, precision_pos), weights)
  recall_avg <- weighted.mean(c(recall_neg, recall_pos), weights)
  f_measure_avg <- weighted.mean(c(f_measure_neg, f_measure_pos), weights)
  roc_auc_avg <- weighted.mean(c(roc_auc_neg, roc_auc_pos), weights)
  mcc_avg <- weighted.mean(c(mcc_neg, mcc_pos), weights)
  k_avg <- weighted.mean(c(k_neg, k_pos), weights)

  # column names
  measures <- c("TPR", "FPR", "Precision", "Recall", "F-measure", "ROC", "MCC", "Kappa")

  # row data
  positive_measures <- c(tpr_pos, fpr_pos, precision_pos, recall_pos,
                         f_measure_pos, roc_auc_pos, mcc_pos, k_pos)
  negative_measures <- c(tpr_neg, fpr_neg, precision_neg, recall_neg,
                         f_measure_neg, roc_auc_neg, mcc_neg, k_neg)
  weighted_average <- c(tpr_avg, fpr_avg, precision_avg, recall_avg,
                        f_measure_avg, roc_auc_avg, mcc_avg, k_avg)

  # create dataframe
  result_df <- data.frame(Class_N=negative_measures,
                          Class_Y=positive_measures,
                          Average=weighted_average)
  rownames(result_df) <- measures
  result_df <- round(result_df, 4)

  return(result_df)
}
```